Jacobi Polynomials

Algorithms, Implementations, and Applications

Huaijin Wang

December 21, 2024

Contents

1 Introduction

The *Jacobi polynomials*, denoted by $J_n^{\alpha,\beta}(x)$, are orthogonal with respect to the *Jacobi weight function* $\omega^{\alpha,\beta}(x) := (1-x)^\alpha(1+x)^\beta$ over $I := (-1,1)$, namely,

$$\int_{-1}^{1} J_n^{\alpha,\beta}(x) J_m^{\alpha,\beta}(x)\omega^{\alpha,\beta}(x)dx = \gamma_n^{\alpha,\beta}\delta_{mn},$$

where $\gamma_n^{\alpha,\beta} = \|J_n^{\alpha,\beta}\|^2_{\omega^{\alpha,\beta}}$. It is know that $J_n^{\alpha,\beta}$ is unique under the sense scaled by a constant (see [Shen (2011), Theorem 3.1]). And the $\gamma_n^{\alpha,\beta}$ is known as (see [Shen (2011), (3.109)])

$$\gamma_n^{\alpha,\beta} = \frac{2^{\alpha+\beta+1}\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{(2n+\alpha+\beta+1)n!\Gamma(n+\alpha+\beta+1)}.$$

2 Jacobi Polynomials

For computing the values of $J_n^{\alpha,\beta}$ over any given $x \in [0,1]$, we leverage its three-term recurrence relation (see [Shen (2011), (3.110) and (3.111)]):

$$J_{n+1}^{\alpha,\beta}(x) = \left(a_n^{\alpha,\beta}x - b_n^{\alpha,\beta}\right) J_n^{\alpha,\beta}(x) - c_n^{\alpha,\beta}J_{n-1}^{\alpha,\beta}(x), n \geq 1, \tag{1}$$

with the initial values:

$$J_0^{\alpha,\beta}(x) = 1, \quad J_1^{\alpha,\beta}(x) = \frac{1}{2}(\alpha+\beta+2)x + \frac{1}{2}(\alpha-\beta).$$

Moreover, the coefficients are known:

$$\begin{aligned}
a_n^{\alpha,\beta} &= \frac{(2n+\alpha+\beta+1)(2n+\alpha+\beta+2)}{2(n+1)(n+\alpha+\beta+1)},\\
b_n^{\alpha,\beta} &= \frac{\left(\beta^2-\alpha^2\right)(2n+\alpha+\beta+1)}{2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)},\\
c_n^{\alpha,\beta} &= \frac{(n+\alpha)(n+\beta)(2n+\alpha+\beta+2)}{(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}.
\end{aligned}$$

Algorithm 1 Computation of $J_n^{\alpha,\beta}(x)$.

Require: order n, α, β and x.
  polylst = ones(size(x)); polyn = polylst;
  poly = $0.5 * (\alpha - \beta + (\alpha+\beta+2) * x)$;
  for $k = 2, \ldots, n$ **do**
    Compute $a_{k-1}^{\alpha,\beta}$, $b_{k-1}^{\alpha,\beta}$, and $c_{k-1}^{\alpha,\beta}$;
    polyn $\leftarrow$ $(a_{k-1}^{\alpha,\beta} * x - b_{k-1}^{\alpha,\beta})$ $*$ poly $-$ $c_{k-1}^{\alpha,\beta}$ $*$ polylst;
    polylst $\leftarrow$ poly; poly $\leftarrow$ polyn;
  end for
  return polyn.

It can be observed from Algorithm 1 that the computation of $J_n^{\alpha,\beta}$ is easy and cheap. Due to the derivative relation (see [Shen (2011), (3.101)]):

$$\partial_x^k J_n^{\alpha,\beta}(x) = d_{n,k}^{\alpha,\beta} J_{n-k}^{\alpha+k,\beta+k}(x), \quad n \geq k,$$

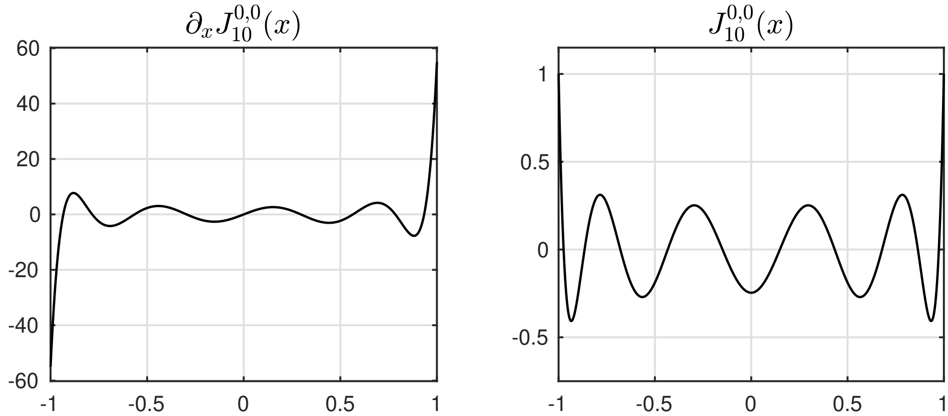
where

$$d_{n,k}^{\alpha,\beta} = \frac{\Gamma(n+k+\alpha+\beta+1)}{2^k\Gamma(n+\alpha+\beta+1)},$$

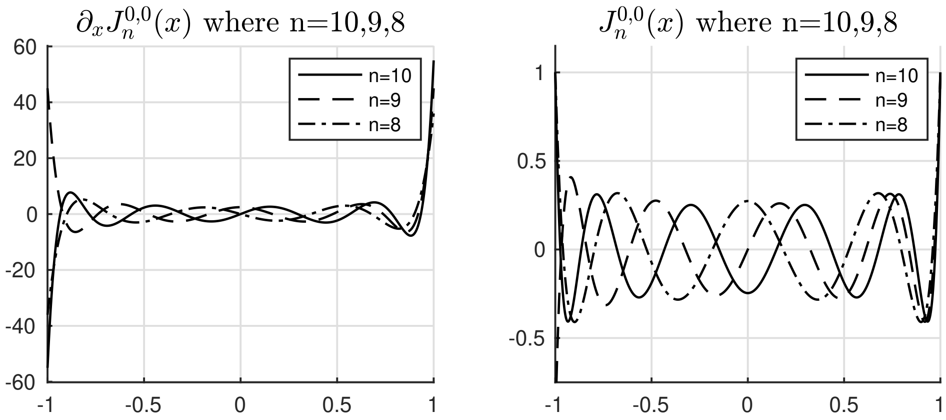
it is straightforward to compute any derivatives of $J_n^{\alpha,\beta}$ via Algorithm 1. And it is also straightforward to compute a string of $J_0^{\alpha,\beta}, \cdots, J_n^{\alpha,\beta}$, with storing results at each step.

There are two .m files, japoly.m and japolym.m, to achieve the procedures. Both of them are developed by Wang Li-Lian on his website: https://blogs.ntu.edu.sg/wanglilian/book/.

```
alp=0; bet=0; n=10; % n indicates the order of polynomials, n=0,1,...
x = linspace(-1,1,1000).';
[dy,y]=japoly(n,alp,bet,x);
subplot(1,2,1), plot(x,dy,'k'), grid on
axis([-1 1 -60.2 60.2]);
title('{$\partial_x J_{10}^{0,0}(x)$}','interpreter','LaTex','FontSize',14);
subplot(1,2,2), plot(x,y,'k'), grid on
axis([-1 1 -.75 1.15]);
title('{$J_{10}^{0,0}(x)$}','interpreter','LaTex','FontSize',14);
```




```
alp=0; bet=0; n=10;
x = linspace(-1,1,1000).';
[DY,Y]=japolym(n,alp,bet,x);
% (k+1)-th row of DY (or Y) saves the value of J_k
% transform it to stack by colums
DY = DY.'; Y = Y.';
%
subplot(1,2,1), hold on, plot(x,DY(:,n+1),'k-','DisplayName', 'n=10'),
plot(x,DY(:,n),'k--','DisplayName', 'n=9'),
plot(x,DY(:,n-1),'k-.','DisplayName', 'n=8'), hold off
legend, grid on
axis([-1 1 -60.2 60.2]);
title(['{$\partial_x J_{n}^{0,0}(x)$}',' where n=10,9,8'],...
    'interpreter','LaTex','FontSize',14);
%
subplot(1,2,2), hold on, plot(x,Y(:,n+1),'k-','DisplayName', 'n=10'),
plot(x,Y(:,n),'k--','DisplayName', 'n=9'),
plot(x,Y(:,n-1),'k-.','DisplayName', 'n=8'), hold off
legend, grid on
axis([-1 1 -.75 1.15]);
title(['{$J_{n}^{0,0}(x)$}',' where n=10,9,8'],...
    'interpreter','LaTex','FontSize',14);
```



3 Jacobi Gauss-type Quadratures

3.1 Jacobi Gauss Quadrature

Let $\{x_j\}_{j=0}^N$ be the zeros of $J_{N+1}^{\alpha,\beta}(x)$. It is known from [Shen (2011), Theorem 3.2] that x_j are real, simple and lie in the interval $(-1,1)$ for $j=0,\cdots,N$. The Lagrange basis polynomials associated with $\{x_j\}_{j=0}^N$ are given by

$$h_j(x) = \prod_{i=0;i\neq j}^{N} \frac{x-x_i}{x_j-x_i}, \quad 0 \le j \le N.$$

Let the *weights*

$$\omega_j = \int_a^b h_j(x)\omega(x)dx, \quad 0 \le j \le N.$$

By [Shen (2011), Theorem 3.5], we know that the nodes $\{x_j\}_{j=0}^N$ and weights $\{\omega_j\}_{j=0}^N$ are Gaussian. That is, the quadrature rule constituted by them satisfies

$$\int_I p(x)\omega^{\alpha,\beta}(x)\mathrm{d}x = \sum_{j=0}^{N} p(x_j)\omega_j, \quad \forall p \in \mathbb{P}_{2N+1},$$

where $\mathbb{P}_{2N+1}$ denotes the collection of polynomials with degree at most $2N+1$. Such quadrature rule are called Gauss quadrature. We know that all weights are positive. And Gauss quadrature shares highest *algebraic accuracy*. In fact, there is no quadrature that have higher algebraic accuracy than Gauss quadrature.

Except for Chebyshev quadrature, other Jacobi quadratures have no closed form and numerical computation is required. A straight and simple way to this end is a method called *eigenvalue method* due to [Golub (1969)]. Let

$$A_{N+1} = \begin{bmatrix} a_0 & \sqrt{b_1} & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{b_{N-1}} & a_{N-1} & \sqrt{b_N} \\ & & & \sqrt{b_N} & a_N \end{bmatrix}, \tag{2}$$

where a_j and b_j are determined by the coefficients of three-term recurrence relation (1):

$$a_j = \frac{b_j^{\alpha,\beta}}{a_j^{\alpha,\beta}} = \frac{\beta^2 - \alpha^2}{(2j+\alpha+\beta)(2j+\alpha+\beta+2)},\ 0 \leqslant j \leqslant N$$

$$b_j = \frac{1}{a_{j-1}^{\alpha,\beta}}\sqrt{\frac{a_{j-1}^{\alpha,\beta}c_j^{\alpha,\beta}}{a_j^{\alpha,\beta}}} = \frac{4j(j+\alpha)(j+\beta)(j+\alpha+\beta)}{(2j+\alpha+\beta-1)(2j+\alpha+\beta)^2(2j+\alpha+\beta+1)},\ 1 \leqslant j \leqslant N.$$

By [Shen (2011), Theorem 3.4 and Theorem 3.6], we have

Theorem 1. *Jacobi Gauss nodes $\{x_j\}_{j=0}^N$ are the eigenvalues of A_{N+1}. Let*

$$\mathbf{Q}(x_j) = \left(Q_0(x_j), Q_1(x_j), \ldots, Q_N(x_j)\right)^T$$

be the orthonormal eigenvector of A_{N+1} corresponding to the eigenvalue x_j, i.e.,

$$A_{N+1}\mathbf{Q}(x_j) = x_j\mathbf{Q}(x_j) \qquad \textit{with} \qquad \mathbf{Q}(x_j)^T\mathbf{Q}(x_j) = 1.$$

Then the Gaussian weights $\{\omega_j\}_{j=0}^N$ can be computed from the first component of the eigenvector $\mathbf{Q}(x_j)$ by using the formula:

$$\omega_j = [Q_0(x_j)]^2\int_{-1}^{1}\omega^{\alpha,\beta}(x)dx = [Q_0(x_j)]^2\frac{2^{\alpha+\beta+1}\Gamma(\alpha+1)\Gamma(\beta+1)}{(\alpha+\beta+1)\Gamma(\alpha+\beta+1)}, \quad 0 \leq j \leq N. \tag{3}$$

Algorithm 2 Computation of Jacobi Gauss Quadrature.

Require: N, α, and β.
  Compute A_{N+1} in (2);
  Determine the eigenpair $\{x_j, \mathbf{Q}(x_j)\}_{j=0}^N$ of A_{N+1};
  Compute weights $\{\omega_j\}_{j=0}^N$ by (3);
  return $N+1$ Jacobi Gauss quadrature $\{x_j, \omega_j\}_{j=0}^N$.

The eigenvalue method is based on the fact that Gaussian nodes correspond to the zeros of orthogonal polynomials, which satisfy a three-term recurrence relation. This approach for computing Gaussian nodes and weights is both classical and stable due to the well-structured nature of A_{N+1}. In most cases, this method is sufficiently accurate. However, its computational cost is $O(N^2)$, and the results may be affected by the condition number of A_{N+1}.

As a remedy, iterative methods, such as Newton's method, are often used with a good initial guess to compute Gaussian nodes, as they are more efficient and accurate. One possible initial guess is given in [Szego (1975), Theorem 8.9.1]. Alternatively, if computational cost is not a concern, the eigenvalues of A_{N+1} can be used as the initial guess to achieve higher accuracy. Once the Gaussian nodes $\{x_j\}_{j=0}^N$ are computed, the Gaussian weights can be determined in following formula (see [Shen (2011), (3.131)]):

$$\omega_j = \frac{\widetilde{G}_N^{\alpha,\beta}}{\left(1-x_j^2\right)\left[\partial_x J_{N+1}^{\alpha,\beta}(x_j)\right]^2},\ 0 \leqslant j \leqslant N, \tag{4}$$

where

$$\widetilde{G}_N^{\alpha,\beta} = \frac{2^{\alpha+\beta+1}\Gamma(N+\alpha+2)\Gamma(N+\beta+2)}{(N+1)!\Gamma(N+\alpha+\beta+2)}.$$

It is evident that with a good initial guess, computing Gaussian nodes and then determining Gaussian weights using (4) may require only $O(N)$ computational cost.

There is a `.m` file, `jags.m`, to compute Jacobi Gauss quadrature, which is developed by Wang Li-Lian on his website: https://blogs.ntu.edu.sg/wanglilian/book/. `jags.m` computes the Gaussian nodes by the eigenvalues of A_{N+1}, and determine the Gaussian weights via (4). It is easy to execute the `jags.m` command as follows:

```
n = 15; alp=1/2; bet=0;
[x,w]=jags(n,alp,bet);
% returns n nodes Gauss quadrature
```

3.2 Jacobi Gauss-Radau Quadrature

Jacobi Gauss-Radau quadrature is a special type of Gauss quadrature with one of its nodes is fixed on the boundary. That is, $\{x_j, \omega_j\}_{j=0}^N$ with $x_0 = -1$ (or $x_N = 1$) satisfy

$$\int_I p(x)\omega^{\alpha,\beta}(x)\mathrm{d}x = \sum_{j=0}^{N} p(x_j)\omega_j, \quad \forall p \in \mathbb{P}_{2N}.$$

Theorem 2 (Theorem 3.26, [Shen (2011)]). *Let $x_0 = -1$ and $\{x_j\}_{j=1}^N$ be the zeros of $J_N^{\alpha,\beta+1}(x)$, and*

$$\omega_0 = \frac{2^{\alpha+\beta+1}(\beta+1)\Gamma^2(\beta+1)N!\Gamma(N+\alpha+1)}{\Gamma(N+\beta+2)\Gamma(N+\alpha+\beta+2)}, \tag{5}$$

$$\omega_j = \frac{1}{(1-x_j)(1+x_j)^2}\frac{\widetilde{G}_{N-1}^{\alpha,\beta+1}}{\left[\partial_x J_N^{\alpha,\beta+1}(x_j)\right]^2}, \quad 1 \le j \le N.$$

where

$$\widetilde{G}_{N-1}^{\alpha,\beta+1} = \frac{2^{\alpha+\beta+2}\Gamma(N+\alpha+1)\Gamma(N+\beta+2)}{N!\Gamma(N+\alpha+\beta+2)}.$$

Then $\{x_j, \omega_j\}_{j=0}^N$ is the desired Jacobi Gauss-Radau quadrature with $x_0 = -1$ fixed.

Similarly, we can show that

Theorem 3. *Let $x_N = 1$ and $\{x_j\}_{j=0}^{N-1}$ be the zeros of $J_N^{\alpha+1,\beta}(x)$, and*

$$\omega_N = \frac{2^{\alpha+\beta+1}(\alpha+1)\Gamma^2(\alpha+1)N!\Gamma(N+\beta+1)}{\Gamma(N+\alpha+2)\Gamma(N+\alpha+\beta+2)}, \tag{6}$$

$$\omega_j = \frac{1}{(1-x_j)^2(1+x_j)}\frac{\widetilde{G}_{N-1}^{\alpha+1,\beta}}{\left[\partial_x J_N^{\alpha+1,\beta}(x_j)\right]^2}, \quad 0 \le j \le N-1.$$

where

$$\widetilde{G}_{N-1}^{\alpha+1,\beta} = \frac{2^{\alpha+\beta+2}\Gamma(N+\beta+1)\Gamma(N+\alpha+2)}{N!\Gamma(N+\alpha+\beta+2)}.$$

Then $\{x_j, \omega_j\}_{j=0}^N$ is the desired Jacobi Gauss-Radau quadrature with $x_N = 1$ fixed.

We denote $\{\xi_{G,N,j}^{\alpha,\beta}, \omega_{G,N,j}^{\alpha,\beta}\}_{j=0}^N$ as the N nodes Gauss quadrature with respect to weight function $\omega^{\alpha,\beta}(x)$, and $\{\xi_{R_l,N,j}^{\alpha,\beta}, \omega_{R_l,N,j}^{\alpha,\beta}\}_{j=0}^N$ the Gauss-Radau quadrature with $\xi_{R_l,N,0}^{\alpha,\beta} = -1$ fixed. Then by [Shen (2011), (3.140)] we have

$$\xi_{R_l,N,j}^{\alpha,\beta} = \xi_{G,N-1,j-1}^{\alpha,\beta+1}, \quad \omega_{R_l,N,j}^{\alpha,\beta} = \frac{\omega_{G,N-1,j-1}^{\alpha,\beta+1}}{1+\xi_{G,N-1,j-1}^{\alpha,\beta+1}}, \quad 1 \le j \le N. \tag{7}$$

Similarly, we denote $\{\xi^{\alpha,\beta}_{R_r,N,j},\omega^{\alpha,\beta}_{R_r,N,j}\}_{j=0}^N$ as the Gauss-Radau quadrature with $\xi^{\alpha,\beta}_{R_r,N,N}=1$ fixed. Then

$$\xi^{\alpha,\beta}_{R_r,N,j}=\xi^{\alpha+1,\beta}_{G,N-1,j},\quad \omega^{\alpha,\beta}_{R_r,N,j}=\frac{\omega^{\alpha+1,\beta}_{G,N-1,j}}{1-\xi^{\alpha+1,\beta}_{G,N-1,j}},\quad 0\le j\le N-1. \tag{8}$$

Both (7) and (8) inform us that the computation of Jacobi Gauss-Radau quadratures can be transformed into computing Jacobi Gauss quadratures.

Algorithm 3 Computation of Jacobi Gauss-Radau Quadrature with $x_0=-1$ fixed.

Require: N, α, and β.
Compute Jacobi Gauss quadrature $\{x_j,\omega_j\}_{j=1}^N$ via Algorithm 2 with $N-1$, α and $\beta+1$;
Let $x_0=-1$, and determine ω_0 by (5);
Compute weights $\omega_j\leftarrow\omega_j/(1+x_j)$ for $j=1,\cdots,N$;
return $N+1$ Jacobi Gauss-Radau quadrature $\{x_j,\omega_j\}_{j=0}^N$.

Algorithm 4 Computation of Jacobi Gauss-Radau Quadrature with $x_N=1$ fixed.

Require: N, α, and β.
Compute Jacobi Gauss quadrature $\{x_j,\omega_j\}_{j=0}^{N-1}$ via Algorithm 2 with $N-1$, $\alpha+1$ and β;
Let $x_N=1$, and determine ω_N by (6);
Compute weights $\omega_j\leftarrow\omega_j/(1-x_j)$ for $j=0,\cdots,N-1$;
return $N+1$ Jacobi Gauss-Radau quadrature $\{x_j,\omega_j\}_{j=0}^N$.

There are two `.m` files, `jagsrd1.m` and `jagsrd2.m`, to compute Jacobi Gauss-Radau quadratures. `jagsrd1.m` computes the Gauss-Radau quadrature with left-end point fixed, while `jagsrd2.m` treats the other case. It is easy to execute the commands as follows:

```
n = 15; alp=1/2; bet=0;
[x,w]=jagsrd1(n,alp,bet); % returns n nodes Gauss-Radau quadrature with x0=-1
[x,w]=jagsrd2(n,alp,bet); % returns n nodes Gauss-Radau quadrature with xN=1
```

3.3 Jacobi Gauss-Lobatto Quadrature

Jacobi Gauss-Lobatto quadrature is a special type of Guass quadrature with two of its nodes are fixed on the boundary. That is $\{x_j,\omega_j\}_{j=0}^N$ with $x_0=-1$ and $x_N=1$ satisfy

$$\int_I p(x)\omega(x)\mathrm{d}x=\sum_{j=0}^N p(x_j)\omega_j,\quad \forall p\in\mathbb{P}_{2N-1}.$$

Theorem 4 (Theorem 3.27 in [Shen (2011)]). *Let $x_0=-1$, $x_N=1$ and $\{x_j\}_{j=1}^{N-1}$ be the zeros of $J^{\alpha+1,\beta+1}_{N-1}(x)$, and let*

$$\begin{aligned}
\omega_0&=\frac{2^{\alpha+\beta+1}(\beta+1)\Gamma^2(\beta+1)\Gamma(N)\Gamma(N+\alpha+1)}{\Gamma(N+\beta+1)\Gamma(N+\alpha+\beta+2)},\\
\omega_N&=\frac{2^{\alpha+\beta+1}(\alpha+1)\Gamma^2(\alpha+1)\Gamma(N)\Gamma(N+\beta+1)}{\Gamma(N+\alpha+1)\Gamma(N+\alpha+\beta+2)},\\
\omega_j&=\frac{1}{\left(1-x_j^2\right)^2}\frac{\widetilde{G}^{\alpha+1,\beta+1}_{N-2}}{\left[\partial_x J^{\alpha+1,\beta+1}_{N-1}(x_j)\right]^2},\quad 1\le j\le N-1,
\end{aligned} \tag{9}$$

where

$$\widetilde{G}^{\alpha+1,\beta+1}_{N-2}=\frac{2^{\alpha+\beta+3}\Gamma(N+\alpha+1)\Gamma(N+\beta+1)}{(N-1)!\Gamma(N+\alpha+\beta+2)}.$$

Then $\{x_j,\omega_j\}_{j=0}^N$ is the desired Jacobi Gauss-Lobatto quadrature with $x_0=-1$ and $x_N=1$ fixed.

We denote $\{\xi_{G,N,j}^{\alpha,\beta},\omega_{G,N,j}^{\alpha,\beta}\}_{j=0}^N$ as the N nodes Gauss quadrature with respect to weight function $\omega^{\alpha,\beta}(x)$, and $\{\xi_{L,N,j}^{\alpha,\beta},\omega_{L,N,j}^{\alpha,\beta}\}_{j=0}^N$ the Gauss-Lobatto quadrature with $\xi_{L,N,0}^{\alpha,\beta}=-1$ and $\xi_{L,N,N}^{\alpha,\beta}=1$ fixed. Then by [Shen (2011), (3.141)] we have

$$\xi_{L,N,j}^{\alpha,\beta}=\xi_{G,N-2,j-1}^{\alpha+1,\beta+1},\quad \omega_{L,N,j}^{\alpha,\beta}=\frac{\omega_{G,N-2,j-1}^{\alpha+1,\beta+1}}{1-\left(\xi_{G,N-2,j-1}^{\alpha+1,\beta+1}\right)^2},\quad 1\le j\le N-1. \tag{10}$$

(10) informs us that the computation of Jacobi Gauss-Lobatto quadrature can be implemented via Jacobi Gauss quadrature.

Algorithm 5 Computation of Jacobi Gauss-Lobatto Quadrature.

Require: N, α, and β.

Compute Jacobi Gauss quadrature $\{x_j,\omega_j\}_{j=1}^{N-1}$ via Algorithm 2 with $N-2$, $\alpha+1$ and $\beta+1$;
Let $x_0=-1$ and $x_N=1$, and determine ω_0 and ω_N by (9);
Compute weights $\omega_j\leftarrow\omega_j/(1-x_j^2)$ for $j=1,\cdots,N-1$;
return $N+1$ Jacobi Gauss-Lobatto quadrature $\{x_j,\omega_j\}_{j=0}^N$.

There is a `.m` file, `jagslb.m`, to compute Jacobi Gauss-Lobatto quadrature, which is developed by Wang Li-Lian on his website: https://blogs.ntu.edu.sg/wanglilian/book/. It is easy to execute the commands as follows:

```
n = 15; alp=1/2; bet=0;
[x,w]=jagslb(n,alp,bet); % returns n nodes Gauss-Lobatto quadrature
```

4 Jacobi Spectral Method

5 Reproduction: [Hou (2019)]

References

[Shen (2011)] Shen J, Tang T, Wang L L. Spectral methods: algorithms, analysis and applications[M]. Springer Science & Business Media, 2011.

[Golub (1969)] Golub G H, Welsch J H. Calculation of Gauss quadrature rules[J]. Mathematics of computation, 1969, 23(106): 221-230.

[Szego (1975)] Szego G. Orthogonal polynomials[M]. American Mathematical Soc., 1975.

[Hou (2019)] Hou D, Lin Y, Azaiez M, et al. A Müntz-collocation spectral method for weakly singular Volterra integral equations[J]. Journal of scientific computing, 2019, 81(3): 2162-2187.